Electronic Version
Stylesheet Version v1.1.1

Description

METHOD AND SYSTEM OF EFFICIENT PACKET REORDERING

BACKGROUND OF INVENTION

[0001] *Field of the Invention*

[0002] The present invention generally relates to the implementation of computer networks, and more particularly, to a method and system for reordering packets received from a high speed data channel.

[0003] *Background Description*

[0004] In a network system, a network handler receives packets from a high-speed network, performs routing actions and then passes the processed packet to an output port for delivery to its intended destination. Packet processing is performed in accordance with one or more specific network protocols. When packets are received out of order, some protocols and classes of service, for example, Fibre Channel Class 2, require delivery of packets in proper se-

quence. The protocol handler provides a mechanism to establish the proper order of packets and to deliver them in the sequential order.

[0005] A straightforward technique for reordering packets is to use a packet table. For each packet received a table entry is generated containing the "sequence count" and the address pointing to the memory area where the packet resides. The sequence count specifies the sequential position of the packet within its flow. Once all packets from the flow are received packets are sorted by sequence count. This approach is inefficient in terms of both memory usage, because of a possibly large number of packets in a flow, and processing time required for searching and sorting the entries. In addition, sorting is only possible when all packets have been received, requiring possibly high buffering resources.

[0006] Another approach is to use a reorder table with depth "n". Similar to the approach described above, for each received packet there is created an entry in the reorder table containing the sequence count and memory address of the packet. The table contains "n" records of the "n" most recently received packets. Once the table is full, the packet with the lowest sequence count is selected for transmis-

sion. This entry is removed making space for the next incoming packet. This approach does not guarantee that packets will always be reordered in that packet reordering is possible only if packets are not displaced for more positions than "n" entries in the table. For example, for table size 8, and for packets with sequence count starting at 20 and up received immediately after the packet with sequence count 10 (received packet sequence is "9, 10, 20, 21, 22, 23, 24, 25, 26, 27, 11 ..."), packets cannot be properly ordered and delivered in sequence.

[0007] In yet another approach, a table entry is recorded only if the packet is received out of sequence. In addition to the sequence count of the last frame received in sequence and memory address of the first packet in that sequence, the table entry contains also the length of data received in order, i.e., the sum of data of all packets from that sub-sequence. For example, for packet sequence "10, 20, 21, 22, 23, 11 ", for the sub-sequence 20 to 23, an entry is created containing the sequence count of the last packet in the sub-sequence (23), the address pointing to the first packet in the sub-sequence (20), and the length of all data contained in packets 20 to 23.

[0008] Proper order of packets is determined by searching for the

table entry with the lowest sequence count and accessing the data of given length from the given memory location. This approach requires large contiguous memory area to store all packets received in sequence, which can be significant. In addition, packets from various flows are received intermixed, so for each flow a separate memory area has to be provided to store all packets received in sequence for that flow. As it is not known in advance how many packets are in a flow and how many packets are in sequence, this technique leads to extremely inefficient memory utilization.

Summary of Invention

[0009] In an aspect of the invention, a method is provided for reordering data packets received out of order comprising the steps of reading context information from a received data packet to determine if it is in a given sequence, comparing said context information to an expected sequence count for the given sequence, and if there is a match storing the received packet with said context information in a memory as a linked list, wherein all packets are in order. The method further provides for creating a new linked list each time a packet is received out-of-order and linking all subsequent packets received in order, constructing a re-

order table of addresses of the first packet for all linked lists and reading packets out of the memory in an order specified by the reorder table.

[0010] In another aspect of the invention, a method for ordering packets is provided comprising the steps of detecting at least one of an in-sequence and an out-of-sequence packet chain in one or more flows, storing at least one of the in-sequence and the out-of-sequence packet chain in a memory, and providing a sequence number with each of the stored in-sequence and the out-of-sequence packet chain. Further provided are the steps of associating the sequence number with an address in the memory of at least one of the stored in-sequence and the out-of-sequence packet chain, and ordering the at least one of the in-sequence and the out-of-sequence packet chain from the memory based on the associated sequence number to provide one or more packet flows all in-sequence.

[0011] In another aspect of the invention, a computer program product is provided comprising a computer usable medium having readable program code embodied in the medium and includes a first computer program code to detect at least one of an in-sequence and an out-of-sequence packet chain in one or more packet flows, a

second computer program code to store the detected at least one of the in-sequence and the out-of-sequence packet chain in a memory, a third computer program code to provide a sequence number with each of the stored in-sequence and the out-of-sequence packet chain. Further provided are a fourth program code to associate the sequence number with an address in the memory of at least one of the stored in-sequence and the out-of-sequence packet chain and a fifth program code to order the at least one of the in-sequence and the out-of-sequence packet chain from the memory based on the associated sequence number to provide one or more packet flows all in-sequence.

Brief Description of Drawings

[0012] The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of embodiments of the invention with reference to the drawings, in which:

[0013] Figure 1 is an illustrative diagram showing an embodiment of a received packet flow received in-order, according to the invention;

[0014] Figure 2 is an illustrative diagram showing an embodiment of a received packet flow received out-of-order, ac-

cording to the invention;

[0015] Figure 3 is illustrative diagram of an embodiment of the invention showing representative component structures for reordering a plurality of packet flows, according to the invention;

[0016] Figures 4A-4B are flow charts showing steps of using the invention for receiving packets that may be out of order in one or more flows, according to the invention;

[0017] Figure 5 is a flow diagram showing steps of using the invention for ordering packets for transmission, according to the invention; and

[0018] Figure 6 is a flow diagram showing steps of an embodiment of using the invention.

Detailed Description

[0019] This invention is directed to providing a method of reordering packets received across a network in which some packets may arrive out of sequential order. Packet chains that are received out-of-order are linked into one or more separate lists (e.g., a linked-list). When subsequent processing of the received packets takes place, e.g., retransmitting the received packets, the out-of-order packet chains are dynamically inserted into correct sequence. This method may be used to track and reorder

packets from multiple incoming flows so that proper order results for each respective flow when packets are re-transmitted (or alternatively, processed in a similar manner). The method of the invention provides for efficient memory utilization and efficient processing during packet processing.

[0020] According to an embodiment of the invention, for every flow, context control areas and linked-list structures are maintained for in-sequence packets and out-of-sequence packets. In addition, the packets received out of sequence are recorded in a reorder table, which is a list of all out-of-order packet segments including the sequence count of each packet and the starting address of each segment. All packets from a single flow received in order are stored in memory as a linked list. In an embodiment, a linked list is used to store the pointer to the next packet in the sequence together with the packet data and packet status information. The last packet in the sequence terminates the linked list by placing this pointer to null, or some other terminator value.

[0021] The sequence count of each received packet is checked against an expected sequence count for that flow. The expected sequence count is typically the sequence count of

the previously received packet incremented by one. Thus, an entry in the reorder table is made for each packet whose sequence count does not match the expected sequence count, and for the first packet received in a flow.

[0022] Figure 1 is an illustrative diagram showing an embodiment of a received packet flow received in-order, generally denoted as reference numeral 100, and a corresponding in-order linked list 110 where the packets are stored pending further processing and/or retransmission. The numeral in each packet of the packet flow (e.g., 0-5) represents a packet sequence number, typically provided by a transmitting entity and may include a timestamp. For situations when a packet flow is received entirely in-order, a reorder table 120 has only one entry reflecting that packet 0 (i.e., the first received packet) and may be found at address location 1 (or other appropriate location). Each subsequent packet, i.e., 1, 2, etc., is linked in order in the linked-list 110 and terminated with a null (or equivalent).

[0023] An expected sequence count 125, at least one per flow, is maintained to track the next expected sequence number (or alternatively, in embodiments, it may also track the last sequence number received). A context control area 130 is also maintained to manage context swaps when

switching between protocol flows and includes context blocks (e.g., 130a or 130b, etc.) for each flow and maintains information associated with every flow, such as, for example, the location of the reorder tables (e.g., 120, etc.) and associated link-list locations (e.g., 110, etc.). In embodiments, the expected sequence count 125 and transmitted packet count 135 may be included as part of the context control area 130, one for every expected flow.

[0024] Figure 2 is an illustrative diagram showing an embodiment of a received packet flow received out-of-order, generally denoted as 150. This exemplary out-of-order packet flow sequence is shown as 0, 1, 5, 2, 3, 4. In addition to the first received packet, the reorder table records those segments of packets (i.e., packet chains, where a chain includes one or more packets in order and may be just one packet) that are out-of-order from the last previously received packet. By way of example, as packets are received, the first packet with sequence number "0"is entered into the reorder table 120 at location 120a with the address of the associated packet address "addr1", 120b. The address, "addr.1", 120b, is the beginning of the linked-list, generally shown as 155, containing received in-sequence segment of packets 0 and 1.

[0025] Similarly, the next out-of-sequence packet is shown as 120c containing packet sequence 5 and can be found at memory location "addr.2", 120d, with the corresponding linked-list, creating a single packet chain, shown as reference numeral 160. Since the next packet following packet 5 is packet 2, packet 2 is considered out-of-sequence. Packet 2 is entered into the reorder table at 120e with "addr.3", 120f. Since the packets 3 and 4 follow in order after packet 2, they are linked into the linked-list following packet 2 creating a three deep packet chain, as shown generally as 165. Any further out-of-order segments may also have entries in the reorder table 120 and have associated linked-list chains similar to 165. Typically, every flow has an associated reorder table 120, expected sequence count 125, transmitted packet count 135, and associated linked-lists (e.g., 155, 160, 165, etc.). As sequences are entered into a reorder table, e.g., 120, an entry is also made into a transmission queue (also know as a transmit queue) such as 230 (Figure 3).

[0026] Figure 3 is an illustrative diagram of an embodiment of the invention showing representative component structures for reordering a plurality of packet flows, generally shown as reference numeral 200. Packets from two flows,

A and B, are received intermixed and portions being out-of-order. Packets associated with flow A are labeled "A", packets associated with flow B are labeled "B". For each of the flows, packet sequence breaks are detected and recorded into the corresponding reorder table, e.g., 215 or 225, and packets are stored in the memory as linked lists, e.g., 210 or 220. For the example of Figure 3, six linked lists are used as there are six distinct out-of-order packet segments, three each from two flows, A and B. Any number of out-of-sequence segments may occur for a flow. Included in this diagram is a resulting transmitted packet stream, generally shown as reference numeral 235, with packets in order for both flows A and B produced as a result of using this invention. A relative sequencing relationship is maintained in the transmitted packet stream 235 between the two packet flows A and B, related to the original arrival order of out-of-sequence packets.

[0027] Figures 4A -6 are flow diagrams showing steps of using the invention. Figures 4A-6 may equally represent a high-level block diagram of components of the invention implementing the steps thereof. The steps of Figures 4-6 may be implemented on computer program code in combination with the appropriate hardware. This computer

program code may be stored on storage media such as a diskette, hard disk, CD-ROM, DVD-ROM or tape, as well as a memory storage device or collection of memory storage devices such as read-only memory (ROM) or random access memory (RAM). Additionally, the computer program code can be transferred to a workstation over the Internet or some other type of network.

[0028] Figures 4A and 4B are flow charts showing steps of using the invention for receiving packets that may be out of order in one or more flows (e.g., 200), starting at 300 where an expected sequence count is initialized to zero (or other beginning expected sequence count). At step 305, a check is made whether a packet has been received, and if not, continual checks are made until one is received. The step 310 may be omitted if the following steps are performed during the time when the received packet resides in the receiving local buffering. At step 310, a received packet is stored into a memory location, often by performing a direct memory access (DMA) transfer to an assigned memory area, by using a pointer to the memory area where the packet is to be stored. At step 315, the header of the packet, which contains information (i.e., packet context) about flow, sequence, and connection, is loaded. This in-

formation may also be obtained from a part of the packet payload, depending on the protocol involved.

[0029] At step 320, the context information is checked against the previously received packet. If the context of the received packet is not the same as the context of the preceding packet, at step 325, the correct context information is loaded from the context control area 130 and the expected sequence count 125 is updated accordingly (i.e., a context swap is performed). The context information is stored in a context status and control block (e.g., 130a or 130b), typically, one context control block per packet flow. If the context information is the same at step 320, processing continues at step 330.

[0030] At step 330, a check is made to see if the received packet is the first in a flow. If it is the first in a flow, processing continues at step 355. If not the first in a flow, then at step 335, the sequence count of the received packet is then compared to the expected sequence count 125 for the flow. If these are equal, the packet is received "in order", and at step 340, the packet is linked to the previously received packet forming a linked list packet chain (e.g., 210a or 210c, etc.). Processing continues at step 360. If, however, at step 335, the packet is received out-

of-order, then at step 345, a new entry is recorded in the reorder table (e.g., 210a, 210b, or 210c). The entry contains the received sequence count (e.g., 215a, etc.) and the pointer (e.g., 215b, etc.) to the memory area where the packet is stored. At step 350, the expected sequence number 125 is updated to the received packet sequence number, and continues at step 355. At step 355, an entry is made into the transmission queue 230 to register a new chain of packets with a flow indicator (e.g., A or B), a beginning sequence number and an associated memory address, respectively. In embodiments, the relative positioning of elements in any of these tables may be in any practical order. At step 360, the expected sequence count is incremented and the context block 130a or 130b information is updated accordingly. The process continues when a new packet is received at 305 or stops when the network device (or equivalent) is reset or powered off.

[0031] Figure 5 is a flow diagram showing steps of using the invention for ordering packets for transmission, starting at step 400, and using the example of Figure 3.

[0032] After reception of a packet flow, and construction of the various structures (e.g., 210, 215, 220, 225, 230, etc), packets from a flow are subsequently delivered in order to

a host or to some other destination according to the invention. To re-order packets for guaranteeing proper sequencing, at step 400, the transmission queue 230 is checked for the first packet chain waiting for transmission. This transmission queue 230 may be built by the steps of Figures 4A and 4B. Packets are subsequently chained together, in order, based upon this transmission queue 230 to create a resulting one or more packet chains, e.g., 235. According to the example of Figure 3, the transmission queue 230 has six entries, 230a-230f, and each entry includes a chain indicator (e.g., "A", or "B"), a sequence count of the beginning packet of the chain, and an address for the first packet in the chain, respectively.

[0033] If there is a packet chain waiting, processing continues with step 405, otherwise checks are continued to be made for a packet chain waiting for transmission. At step 405, a next entry (or first entry if a first access) from the transmission table 230 is accessed and the flow context information is accessed by using the context block (e.g., 130a or 130b) associated with the flow indicator (e.g., A) to determine where the first packet of the packet chain is stored (for the first packet chain, this may be entry 230a

of the transmission queue).

[0034] At step 410, the context information of this packet is checked. A check is made at step 415 to determine whether the context of the flow to which this packet belongs is the same as the previous packet chain and whether a context switch is required. If a context switch is required, at step 420, the proper context information is loaded. Processing continues at step 425 where using the associated context block for the flow (e.g., 130a or 130b), the reorder table (e.g., 215 or 225) belonging to this flow is searched for the lowest sequence count (e.g., 215a). At step 425, the lowest sequence count from the reorder table (e.g., 215 or 225) is checked if this is the next packet which is to be sent. At step 430, this condition is determined by calculating the number of packets from that flow already sent. The number of sent packets from a flow is a difference between the lowest sequence count from the reorder table and the minimal sequence count of that flow (this is recorded in the context block). If this packet chain is not the next to be sent, it is skipped at step 435, and the queue of packet chains waiting for transmission is checked for the next entry.

[0035] If this is the next packet chain to be transmitted (at step

430), then at step 440, all packets from the linked list (e.g., chains 210a, 210b, 210c, or a chain from 220, etc.) are accessed and transferred. At step 445, the number of packets sent (i.e., the transmitted packet count 135, for this flow) is incremented. At step 450, a check is made whether the last packet in the linked list (e.g. 210a, 21b, or 210c) has been processed. If not, at step 455, the next packet in the chain is accessed and processing continues with step 440.

[0036] If all the packets in the chain are processed from the linked list, the entry in the reorder table (e.g., 215 or 225), and from the transmission queue 230, belonging to this chain are removed. At step 465, the context block is updated to reflect this transaction. Processing continues at step 400.

[0037] Further applying the steps of Figure 5 to the example of Figure 3, generally, all entries of the transmission queue are processed in turn. The next entry (230b) in the transmission queue in this example has a sequence count 0 of the flow B at the address 2. As this is the first packet from the flow B, all three packets from this chain are sent. The context block belonging to the flow B is updated to record 3 sent packets and the entry (230b) in the reorder table B

is removed.

[0038] The next entry (i.e., 230c) in the transmission queue in this example is the sequence count 5 of the flow A. The reorder table for flow A (i.e., 215) is searched for the lowest sequence count, which is by this iteration, sequence count 2. Thus, the packet chain from flow A starting with sequence count 5 is skipped, and the next entry from the transmission queue is checked. In this example, the next entry (230d) in the transmission queue is the sequence count 2 of the flow A at the address 4. As this is the next packet to be sent from the flow A, all three packets from this chain are sent. The context block (e.g., 130a) belonging to the flow A is updated to record a total of 5 sent packets and the entry (2, addr. 4) in the reorder table A (215) is removed. This process repeats for the remaining entries (i.e., 230e and 230f) until all packets from the transmission queue 230 are sent. All packets from flows A and B are transmitted in order, but the flows are intermixed with a relative relationship to the order of segments received.

[0039] Figure 6 is a flow diagram showing the steps of using the invention in an embodiment, beginning at step 500, where at least one of an in-sequence or an out-

of-sequence packet chain in the one or more flows. At step 510, storing at least one of the in-sequence or out of sequence packet chain in a memory is done. At step 520, a sequence number is provided with each of the stored in-sequence and out-of-sequence packet chains. At step 530, the sequence number is associated with an address in the memory of at least one of the stored in sequence or out-of-sequence packet chain. At step 540, the at least one of the in-sequence and out-of sequence packet chain are retrieved and ordered from the memory based on the associated sequence number to provide one or more flows having all in-sequence packets.

[0040] This invention also may be implemented as a multiprocessor or multithreaded implementation. For a multiprocessor/multithread environment, in an embodiment, multiple flows are processed on different processors, and all packet chains are queued for transmission to an outbound line. Processing packets from the same flow on several processors can bring packets out of order due to different processing latencies associated to each packet. The invention guarantees transmission of all packets from all flows in order. For multiprocessor environments, at transmission, data may be sorted by searching the reorder ta-

ble depending on the flow identification and sequence counts of each particular flow. Only packets in a particular flow should be properly sorted, not the packets between various flows.

[0041] This invention may use one or more pre-allocated fixed sized buffers for storing received packets. This allows for simple memory management, eliminating the need for complex and time consuming dynamic memory management tasks, like garbage collection. This results in a more efficient memory management system. No assumptions are made concerning the size of the buffers, allowing for implementation of multiple sized buffers, or independent linked lists of small and large buffers for storing appropriately sized packets (e.g., small or large packets for flows A and B, respectively). The disclosed invention is applicable to all these variations of embodiments.

[0042] The invention eliminates the need to pre-allocate significant contiguous memory area for each flow received and for each sub sequence of packets received in order. As it is not known in advance how many packets are contained in a single flow and how many of those are going to be received in order, the invention provides for efficient memory utilization and processing overhead.

[0043] Additionally, the invention may use the reorder table of relatively small size, as only points of breaks in the flow sequence are recorded. This results in short processing time and low memory requirements.

[0044] While the invention has been described in terms of embodiments, those skilled in the art will recognize that the invention can be practiced with modifications and in the spirit and scope of the appended claims.